

A nonenumerative algorithm to find the k longest (shortest) paths in a DAG

Fatih Koçan

Abstract

In this paper, we present a novel and efficient algorithm to find the k longest (shortest) paths between sources and sinks in a directed acyclic graph (DAG). The algorithm does not enumerate paths therefore it is especially useful for very large k values. It is based on the Valued-Sum-of-Product (VSOP) tool, which is an extension of Zero-suppressed Binary Decision Diagrams (ZBDDs). We assessed the performance of this algorithm with a DAG model of a path-intensive combinational circuit, viz. c6288, that has $\sim 10^{20}$ paths. We found that it took about 64 minutes to compute all paths in this DAG along with their lengths.

1 Introduction

The classical problem of finding the shortest or longest paths in a directed graph has been generalized to find the k shortest or k longest paths. In the k shortest path problem, the paths are ranked in the increasing order of their lengths while in the k longest path problem, the paths are ranked in the decreasing order of their lengths. The k shortest (longest) path problem is to find k shortest (longest) paths between two given vertices s and t in a directed graph with non-negative edge weights for a positive k value. The shortest (longest) paths can be simple or with loops. There are two well-known k shortest nonsimple path finding algorithm. One of them achieves $O(m + kn \log n)$ [4] and the second one achieves $O(m + n \log n + k)$ [3] run times.

The problem of finding the k shortest simple paths has been studied in [5, 6, 12, 17, 18, 1, 2]. The best asymptotic performance has been obtained in [17, 18] and it achieves $O(kn(m + n \log n))$ worst-case run time. The work in [5] outperforms the algorithms with the best known results, esp. for large graphs. It can be noticed that when k is very large, such as $k = 10^{15}$, the algorithms become impractical.

There are also studies to compute the k longest paths in DAGs [13, 19, 9]. All algorithms enumerate paths in the order of their lengths. In this paper, we focus on finding k longest paths in directed acyclic graph (DAG) with any integer weights. Later, we modify the algorithm to compute the k shortest paths, the $k - th$ longest path, and the $k - th$ shortest path. Our algorithm does

not enumerate paths and relies on the implicit calculation of paths and their lengths. Also, we aim at very large k values.

Our algorithm is inspired by the recent usage of the ZBDD tool in path delay fault coverage (PDF) calculation [16, 10, 11] and the availability of the Valued-Sum-of-Products (VSOP) tool [15], an extension to the ZBDD tool [7]. The algorithms are all *nonenumerative* algorithms. The algorithms may encounter memory overflow during their computations. In this case, the algorithms would be run on partitioned graphs to avoid memory overflow that causes the memory swapping, which in turn increase the run times [11].

The rest of the paper is organized as follows. Section II overviews the VSOP tool. Section III introduces the k longest (shortest) path finding algorithm. Section IV gives the experimental result for a path-intensive DAG. Finally, we conclude the paper.

2 Valued-Sum-of-Product (VSOP) Tool based on ZBDDs

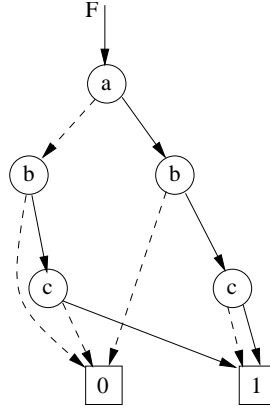
A *combinatorial item set* is a set of elements each of which is a combination out of n items. Zero-suppressed BDDs (ZBDDs) are special type of BDDs that are designed for implicit representation and efficient manipulation of combinatorial item sets [7]. The manipulation time is proportional to the size of the underlying ZBDD. Therefore, ZBDDs utilize certain reduction rules to compactly represent a set. The size of a ZBDD is also sensitive to the order of items (i.e., variables) in a ZBDD. Many static and dynamic variable ordering approaches are investigated for that purpose. We explain a ZBDD with the help of an example.

Let $F = \{abc, ab, bc\}$ be a combinatorial item set. The set of items in F are $\{a, b, c\}$. Figure 1 illustrates the ZBDD for F . The items are ordered as a being the first, b being the second, and c being the third. There are two types of edges: 0-edges (dashed line) and 1-edge (solid line). Each path from the root to leaf 1 corresponds to a combination in the set. In a path, an solid (dashed) outgoing edge from a node indicates that the respective item is included (excluded) in (from) the combination.

Valued-Sum-of-Product (VSOP) [15, 14] is an extension to ZBDD, and supports assignment of values to the terms in a sum-of-products expression and enables efficient manipulation of expressions. For example, $E = 2ab + ac - 3bd$ is a VSOP expression where terms ab , ac , and bd have 2, 1, and -3 values, respectively. In VSOP algebra, addition and subtraction follow the ordinary rule ($a + a = 2a$) but the multiplication does not, and $a \times a = a$, not a^2 . VSOP adopts the base (-2) binary encoding of numbers and an integer number is decomposed into an n -digit vector of ZBDD $\{F_0, F_1, \dots, F_{n-1}\}$. F_i is a ZBDD and stores the terms which have a 1 in their $(i+1)^{th}$ digits in their encodings. Specifically, F_0 is the set of odd-valued terms, F_1 is the set of terms which have a 1 in the second position etc.

The VSOP expression $F = 4abc + 5ab + 3bc + a$ is represented in VSOP as in Table 1. The encoding yields $F_0 = \{ab, bc, a\}$, $F_1 = \{bc\}$, and $F_2 = \{abc, ab, bc\}$.

The following VSOP operations are used in the following sections and their

Fig. 1: ZBDD for $F = \{abc, ab, bc\}$ Tab. 1: Encoding values of F 's terms

F	Value	F_2	F_1	F_0
abc	4(100)	1	0	0
ab	5(101)	1	0	1
bc	3(111)	1	1	1
a	1(001)	0	0	1

details can be found in [14].

- $E_1.Permut(E_2)$: Extract terms in E_1 each of which is included in one of combinations in E_2 .
- $E_1.Restrict(E_2)$: Extract terms in E_1 each of which includes one of combinations in E_2 .
- $E_1.TermsOP(E_2)$: Filter terms in E_1 "OP" to the constant term of E_2 where $OP = \{EQ, NE, LE, LT, GT, GE\}$.
- $E_1.OP_Const(K)$: Filter terms in E_1 whose terms are "OP" to constant K where $OP = \{EQ, NE, LE, LT, GT, GE\}$.
- $E_1 op E_2$: Two expressions are subject to arithmetic or logical operation op , where $OP = \{+, -, *, /, \%, ==, !=, <, <=, >, >=\}$.
- $E.CountTerms()$: The number of minterms in expression E .
- $E.TotalVal()$, $E.MaxVal()$, $E.MinVal()$: Sum of the minterm values, maximum value in the set, and minimum value in the set.
- $I.GetInt()$: Convert VSOP value I to integer.
- $E.MinCover()$: Return a minimum-valued term in E .

Tab. 2: VSOP examples

Operation	Result
$F.Restrict(a)$	$4abc + 5ab + a$
$F.Restrict(ab)$	$4abc + 5ab$
$F.Restrict(a + b)$	$4abc + 5ab + 3bc + a$
$F.Permut(ab)$	$5ab + a$
$F.Permut(abc)$	$4abc + 5ab + 3bc + a$
$F.Permut(c)$	0
$F.CountTerms()$	4
$F.MaxVal()$	5
$F.MinVal()$	1
$F.TermsGE(3)$	$4abc + 5ab + 3bc$
$F.TermsLT(3)$	a
$F + G$	$4abc + 10ab + a$
$F - G$	$4abc + 6bc + a$
$F \times G$	$5abc + 30ab - 9bc$
$F == G$	ab
$F > G$	$abc + bc + a$
$G > F$	0
$F! = G$	$abc + bc + a$
$F.MinCover()$	a
$F.MaxCover()$	$5ab$

- $E.MaxCover()$: Return a maximum-valued term in E .

Let $F = 4abc + 5ab + 3bc + a$ and $G = 5ab - 3bc$. Some operations on F and G and their outcomes are tabulated in Table 2. Comparison operations are better understood if we assume 0 coefficients for the nonexisting terms in G i.e., $G = 0abc + 5ab - 3bc + 0a$.

3 Algorithm

This section introduces the algorithm that store all or selected paths along with their lengths. Then, we select the k longest paths from this path database. Later, we explain the k shortest path finding algorithm.

To find the k longest paths, we utilize a data-driven binary search (DDBS) algorithm (Algo. 1). Since sorting and index-based accessing to the paths in a VSOP is not possible, the DDBS algorithm uses nonenumerative VSOP operators to find the set of k longest paths from the built path database. Also, in this algorithm, when $\lfloor \frac{max+min}{min} \rfloor = min$, the search to find the set is repeated for $min + 1$ in the last step. Otherwise, the algorithm would not terminate.

We present an algorithm (Algo. 2) to find the k longest paths with VSOP tool nonenumeratively. Algorithm 2 starts from the source vertices and inductively builds the path database in the topological order of vertices. At each

Algorithm 1 $TopK(K, \mathcal{L})$

Require: $0 < K \leq |\mathcal{L}|$
 $min \leftarrow \mathcal{L}.MinVal()$
 $max \leftarrow \mathcal{L}.MaxVal()$
 $mid_prev \leftarrow 0$
while !done **do**
 $mid_prev \leftarrow mid$
 $mid \leftarrow (min + max)/2$
 if $CtoI_EQ(mid, mid_prev).GetInt()$ **then**
 $mid \leftarrow mid + 1$
 done $\leftarrow 1$
 break
 end if
 $c1 \leftarrow (\mathcal{L}.EQ_Const(mid)).CountTerms()$
 $c2 \leftarrow (\mathcal{L}.GT_Const(mid)).CountTerms()$
 $c3 \leftarrow c1 + c2$
 $flag \leftarrow c3.GT_Const(K) \ \&\& \ c2.LT_Const(K)$
 if $(c3.EQ_Const(K)) || flag$ **then**
 done $\leftarrow 1$
 break
 else if $(c3.LT_Const(K)).GetInt()$ **then**
 $max \leftarrow mid$
 continue
 else if $(c3.GT_Const(K)).GetInt()$ **then**
 $min \leftarrow mid$
 continue
 end if
end while
return $\mathcal{L}.FilterThen(\mathcal{L}.GE_Const(mid))$

vertex, the algorithm builds the set of partial paths that end at this node. The sum of sink nodes' paths is the set of all paths along with their lengths. In the algorithm, we optionally utilize early pruning of infeasible paths by calling $TopK()$ algorithm with parameter k .

In Algo. 2, \mathcal{L} and \mathcal{L}_i s are VSOP expressions, and $\mathcal{L}_i == \mathcal{L}_i$ removes the values of terms in a VSOP expression. For example, $((2ab + 3bc) == (2ab + 3bc)) \Rightarrow ab + bc$. The last loop takes the union of all paths that end at the sinks. After that path queries can be performed on all paths using VSOP operations. In our algorithm, we query the k longest paths. Inductive building of a set of paths is illustrated with the example below.

Example: In Fig. 2, $\mathcal{L}_2 = 0$, and $\mathcal{L}_3 = 0$. $c_{i,j}$ is the length associated with edge $e_{i,j}$. Thus, the partial paths at v_7 along with their lengths are computed as follows. We initialize $\mathcal{L}_7 = 0$. Since v_2 is a source vertex, $\mathcal{L}_7 = \mathcal{L}_7 + 4 \cdot v_2$. Since v_3 is a source vertex, $\mathcal{L}_7 = \mathcal{L}_7 + 4 \cdot v_3$. After that $\mathcal{L}_7 = \mathcal{L}_7 \cdot v_7 = 4 \cdot v_2 \cdot v_7 + 4 \cdot v_3 \cdot v_7$.

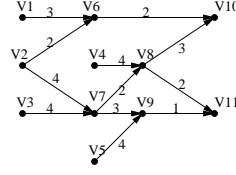


Fig. 2: A DAG with edge costs.

We compute \mathcal{L}_8 as follows. Since v_4 is a source, $\mathcal{L}_4 = 0$. Thus, $\mathcal{L}_8 = \mathcal{L}_8 + 4 \cdot v_4$ and $\mathcal{L}_8 = \mathcal{L}_8 + \mathcal{L}_7 + (v_2 \cdot v_7 + v_3 \cdot v_7) \cdot 2 = 4 \cdot v_4 + 6 \cdot v_2 \cdot v_7 + 6 \cdot v_3 \cdot v_7$. Finally, $\mathcal{L}_8 = \mathcal{L}_8 \cdot v_8 = 4 \cdot v_4 \cdot v_8 + 6 \cdot v_2 \cdot v_7 \cdot v_8 + 6 \cdot v_3 \cdot v_7 \cdot v_8$. We compute the partial paths at each vertex in topological order similarly for all vertices.

Algorithm 2 Paths and their lengths: **K-longest()**

Require: $V[1..N]$: Topologically sorted vertices

```

for  $i \leftarrow 1$ ;  $i \leq N$ ;  $i \leftarrow i + 1$  do
   $\mathcal{L}_i \leftarrow 0$ 
  for each incidence vertex  $v_j$  of  $v_i$  do
    if  $v_j$  is a source then
       $\mathcal{L}_i \leftarrow \mathcal{L}_i + c_{i,j} \cdot v_j$ 
    else
       $\mathcal{L}_i \leftarrow \mathcal{L}_i + \mathcal{L}_j + c_{i,j} \cdot (\mathcal{L}_j == \mathcal{L}_j)$ 
    end if
     $TopK(k, \mathcal{L}_i)$  – early prune code
  end for
   $\mathcal{L}_i \leftarrow \mathcal{L}_i \cdot v_i$ 
end for
 $\mathcal{L} \leftarrow 0$ 
for each sink  $v_i$  do
   $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_i$ 
end for
 $longest \leftarrow \mathcal{L}.MaxVal()$ 
return  $TopK(k, \mathcal{L})$ 

```

By calling $TopK()$ algorithm after every partial path set calculation at each vertex we can eliminate infeasible partial paths early from the set of all partial paths at this vertex. A partial path that is not in the top K partial paths is an infeasible partial path. In the final k longest path set, there cannot be a path starting with an infeasible partial path. Therefore, it is safe to remove them from consideration. This early pruning of them would speed up the algorithm.

3.1 Finding the k shortest paths

We modify the $TopK()$ algorithm (Algorithm 1) to find the k shortest paths. All we need to:

1. change GT into LT in calculation of $c2$,
2. change $max \leftarrow mid$ to $min \leftarrow mid$,
3. change $min \leftarrow mid$ to $max \leftarrow mid$, and
4. change GE in the last statement to LE .

The modified algorithm is given in Algorithm 3.

Algorithm 3 $TopK(K, \mathcal{L})$: shortest path version

```

Require:  $0 < K \leq |\mathcal{L}|$ 
 $min \leftarrow \mathcal{L}.MinVal()$ 
 $max \leftarrow \mathcal{L}.MaxVal()$ 
 $mid\_prev \leftarrow 0$ 
while !done do
   $mid\_prev \leftarrow mid$ 
   $mid \leftarrow (min + max)/2$ 
  if  $CtoI\_EQ(mid, mid\_prev).GetInt()$  then
     $mid \leftarrow mid + 1$ 
     $done \leftarrow 1$ 
    break
  end if
   $c1 \leftarrow (\mathcal{L}.EQ\_Const(mid)).CountTerms()$ 
   $c2 \leftarrow (\mathcal{L}.LT\_Const(mid)).CountTerms()$ 
   $c3 \leftarrow c1 + c2$ 
   $flag \leftarrow c3.GT\_Const(K) \&\& c2.LT\_Const(K)$ 
  if  $(c3.EQ\_Const(K)) || flag$  then
     $done \leftarrow 1$ 
    break
  else if  $(c3.LT\_Const(K)).GetInt()$  then
     $min \leftarrow mid$ 
    continue
  else if  $(c3.GT\_Const(K)).GetInt()$  then
     $max \leftarrow mid$ 
    continue
  end if
end while
return  $\mathcal{L}.FilterThen(\mathcal{L}.LE\_Const(mid))$ 

```

Tab. 3: c6288

Top K	Nodes	Time	Mem%	Paths
5×10^5	20592319	6:34	4.5	572,976
10^6	26921861	8:33	4.6	1,485,955
5×10^6	36561672	12:19	8.9	6,561,113
10^7	44147879	15:31	9	16,629,545
All	326301920	64:17	70.6	$\sim 10^{20}$

3.2 Finding the k -th longest and k -shortest path

After we find the k longest paths, we can query the $k - th$ longest path among them. Let \mathcal{Z} be the set of k longest paths. $\mathcal{Z}.minCover()$ would return a minimum length path from the set \mathcal{Z} , which is the $k - th$ longest path.

Similarly, we can find the k shortest paths and query the $k - th$ shortest path among them. Let \mathcal{Z} be the set of k shortest paths. $\mathcal{Z}.maxCover()$ would return a maximum length path from the set \mathcal{Z} , which is the $k - th$ shortest path.

Note that $k - th$ shortest or longest path may not be unique. In that case, $TopK()$ algorithm may return more than K paths as solution set.

4 Experimental Result

We implemented the proposed algorithms and assessed their performances using a DAG model of a path-intensive c6288 circuit benchmark [8]. This benchmark has 32 source and 32 sink vertices with a total of 2448 vertices and 4800 edges. There are 9.89434×10^{19} paths from sources to sinks in this benchmark. Experiment was performed on an Intel-86-based 64-bit processor with two dual 2.66-GHz CPUs, and 20-Gbyte RAM with the Linux operating system.

Table 3 tabulates the results for c6288. The edge weights are set to a number between 1 and 10 that is generated randomly. We calculate the number of used ZBDD nodes, time in minutes:seconds format, percentage of memory usage, and the number of paths in the query of top K paths for various values of K . For example, Algorithm 2 computes top 5×10^5 paths in 6 minutes and 34 seconds with 4.5% of 20 GB memory. The algorithm allocates 20592319 ZBDD nodes and returns 572,976 longest paths. Note that we do not find exactly k paths when the $k - th$ path is not unique.

5 Conclusion

In this paper we introduced a novel algorithm based on VSOP and ZBDD to extract the k longest (shortest) paths and the $k - th$ longest (shortest) path from a DAG. This algorithm outperforms existing algorithm when the value of k is very large since the run-times of the algorithms become impractical for

very large k values. We assessed the performance of our algorithm with a path-intensive DAG that has $\sim 10^{20}$ paths. We were able to compute all paths and their lengths in about 64 minutes.

The proposed algorithm can be used as a front end software to selected critical paths for timing analysis. It can also be used in the path delay fault coverage calculation to enable delay-sensitive coverage calculation.

References

- [1] J. A. Azevedo, M. E. O. Santos Costa, J. J. E. R. Silvestre Madeira, and Ernesto de Queirós Vieira Martins. An algorithm for the ranking of shortest paths. *Eur. J. Operational Research*, 69:97–106, 1993.
- [2] A. W. Brander and Mark C. Sinclair. A comparative study of k -shortest path algorithms. In *Proc. 11th UK Performance Engineering Worksh. for Computer and Telecommunications Systems*, September 1995.
- [3] David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, February 1999.
- [4] B. L. Fox. k -th shortest paths and applications to the probabilistic networks. In *ORSA/TIMS Joint National Mtg.*, volume 23, page B263, 1975.
- [5] John Hersberger, Matthew Maxel, and Subhash Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Trans. Algorithms*, 3(4), November 2007.
- [6] Walter Hoffman and Richard Pavley. A method for the solution of the n th best path problem. *J. ACM*, 6(4):506–514, October 1959.
- [7] Shin ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Design Automation Conference*, pages 272–277, 1993.
- [8] ITC99. <http://www.cad.polito.it/tools/itc99.html>.
- [9] Yun-Cheng Ju and Resve A. Saleh. Incremental techniques for the identification of statically sensitizable critical paths. In *Proceedings of the 28th ACM/IEEE Design Automation Conference, DAC '91*, pages 541–546, New York, NY, USA, 1991. ACM.
- [10] Fatih Kocan and Mehmet Hadi Gunes. On the zbdd-based nonenumerative path delay fault coverage calculation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(7):1137–1143, 2005.
- [11] Fatih Kocan, Lun Li, and Daniel G. Saab. Exact path delay fault coverage calculation of partitioned circuits. *IEEE Trans. on Computers*, 58(6):858–864, 2009.

-
- [12] Eugene L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.
 - [13] Wing-Ning Li, S.M. Reddy, and S.K. Sahni. On path selection in combinational logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 8(1):56–63, jan 1989.
 - [14] Shin-Ichi Minato. VSOP (valued-sum-of-products) calculator based on zero-suppressed BDDs. Technical Report TCS-TR-A-05-3, Hokkaido University, Division of Computer Science, May 2005.
 - [15] Shin-Ichi Minato. VSOP (valued-sum-of-products) calculator for knowledge processing based on zero-suppressed BDDs. In *LNAI*, pages 40–58, 2006.
 - [16] Saravanan Padmanaban, Maria K. Michael, and Spyros Tragoudas. Exact path delay fault coverage with fundamental ZBDD operations. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(3):305–316, 2003.
 - [17] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.
 - [18] J. Y. Yen. Another algorithm for finding the k shortest loopless network paths. In *Proceedings of the 41st Meeting of the Operations Research Society of America*, volume 20, 1972.
 - [19] S.H.C. Yen, D.H.C. Du, and S. Ghanta. Efficient algorithms for extracting the k most critical paths in timing analysis. In *Design Automation, 1989. 26th Conference on*, pages 649 – 654, june 1989.